

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4  #define TOL 0.0000001
5
6  /* Problem # 2 Using Successive OverRelaxation Method
7
8  */
9
10 double dblAbs(double num){
11     num = sqrt(num*num);
12     return num;
13 }
14
15 int main()
16 {
17
18     double a[13][13] = {
19         { 1.0250, -0.4000, 0.0000, 0.0000, -0.6250, 0.0000, 0.0000, 0.0000, 0.0000,
20         0.0000, 0.0000, 0.0000, 0.0000},
21         {-0.4000, 1.9135, -0.2998, 0.0000, 0.0000, 0.0000, -1.0338, -0.1799, 0.0000,
22         0.0000, 0.0000, 0.0000, 0.0000},
23         { 0.0000, -0.2998, 2.1415, -0.4417, 0.0000, 0.0000, -1.4000, 0.0000, 0.0000,
24         0.0000, 0.0000, 0.0000, 0.0000},
25         { 0.0000, 0.0000, -0.4417, 1.0077, 0.0000, 0.0000, 0.0000, 0.0000, -0.5660,
26         0.0000, 0.0000, 0.0000, 0.0000},
27         {-0.6250, 0.0000, 0.0000, 0.0000, 2.0500, -0.8000, 0.0000, 0.0000, 0.0000,
28         -0.6250, 0.0000, 0.0000, 0.0000},
29         { 0.0000, -1.0338, 0.0000, 0.0000, 0.0000, -0.8000, 3.8097, -0.4401, 0.0000,
30         -1.0000, -0.0595, -0.4764, 0.0000},
31         { 0.0000, -0.1799, -1.4000, 0.0000, 0.0000, -0.4401, 4.0492, -0.7067, 0.0000,
32         0.0000, 0.0000, -1.1159, -0.2067},
33         { 0.0000, 0.0000, 0.0000, -0.5660, 0.0000, 0.0000, -0.7067, 2.2161, 0.0000,
34         0.0000, 0.0000, 0.0000, -0.9433},
35         { 0.0000, 0.0000, 0.0000, 0.0000, -0.6250, 0.0000, 0.0000, 0.0000, 1.0250,
36         -0.4000, 0.0000, 0.0000, 0.0000},
37         { 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, -1.0000, 0.0000, 0.0000,
38         -0.4000, 2.0667, -0.6667, 0.0000},
39         { 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, -0.0595, 0.0000, 0.0000, 0.0000,
40         -0.6667, 1.3788, -0.6526, 0.0000},
41         { 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, -0.4764, -1.1159, 0.0000, 0.0000,
42         0.0000, -0.6526, 2.5098, -0.2650},
43         { 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, -0.2067, -0.9433, 0.0000,
44         0.0000, 0.0000, -0.2650, 1.4150},
45     };
46
47     double b[13] = {2, 0, 0, 0, 4, 0, 0, 0, 2, 0, 0, 0, 0};
48
49     double x_old[13] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
50     double x_new[13] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
51     double err[13];
52     double u0 = 100;
53     double w = 0.01;
54     double w0 = 1;
55     double dw = 0.05;
56     double wn = 1.5;
57     double dummy, m;
58     int method = 2; // Methods: 0: Jacobi Method, 1: Gauss-Seidel,
59
60     2: SOR
61
62     int total, col, row, k, i, j;
63     double maxerr, diff, L2;
64
65     total = sizeof(a)/sizeof(a[0][0]); //
66     col = sizeof(a[0])/sizeof(a[0][0]); // Determines the number of rows and columns
67     row = total/col;
68
69     for(i=0; i<row; i++){
70         b[i] = b[i]*u0;
71     }
72
73     //Perform Partial Pivoting;
74     for(i=0; i<row-1; i++){
75         //comparison to select the pivot
76         for(j = i+1; j<row; j++) {
77             if (dblAbs(a[j][i]) > dblAbs(a[i][i])) {
78                 for(k=0; k<col; k++){
79                     dummy = a[i][k];
80                     a[i][k] = a[j][k];
81                     a[j][k] = dummy;
82                 }
83                 dummy = b[i];
84                 b[i] = b[j];
85             }
86         }
87     }

```

```

71         b[j] = dummy;
72     }
73 }
74 //Make an Upper Triangular Matrix
75 for(j = i+1; j<row; j++) {
76     m = a[j][i]/a[i][i];
77     for(k = 0; k<col; k++){
78         a[j][k] = a[j][k] - m*a[i][k];
79     }
80     b[j] = b[j] - m*b[i];
81 }
82 }
83
84 if (method < 2) {
85     wn = w0;
86 }else {
87     printf(" w Total Iterations          \n");
88     printf("\n");
89 }
90
91 for (w = w0; w <=wn+dw; w = w + dw) {
92     maxerr = 100; // So that it will go to while loop
93     k = 0;
94
95     for (i = 0; i < row; i++){
96         x_old[i] = 0;
97         x_new[i] = 0;
98     }
99
100    while (maxerr > TOL) {
101        k++; //Iteration number
102
103        for (i = 0; i < row; i++) {
104            /** Using Jacobi method only **/
105
106            // Update values of x_new
107            x_new[i] = b[i];
108            for (j = 0; j < col; j++) {
109                if( i != j) {
110                    x_new[i] = x_new[i] - a[i][j] * x_old[j];
111                }
112            }
113            x_new[i] = (x_new[i])/a[i][i];
114
115            /** with SOR Method **/
116            if(method == 2) {
117                x_new[i] = w*x_new[i] + (1-w)*x_old[i];
118            }
119            diff = x_new[i] - x_old[i];
120            err[i] = sqrt(diff*diff); // Get absolute value of convergence
121            criterion vectors
122
123            /** with Gauss-Seidel Method **/
124            if (method > 0) {
125                x_old[i] = x_new[i]; // Comment out to use Jacobi method
126            }
127        }
128
129        L2 = 0; // Linfinity - norm
130        for (i = 0; i < row; i++) {
131            x_old[i] = x_new[i]; // Update values of x_old
132            if (err[i] > L2) {
133                L2 = err[i];
134            }
135        }
136        maxerr = L2;
137
138        if (method <2) {
139            printf("Iteration number = %d with tolerance = %0.9lf \n",k,maxerr);
140        }
141    }
142 }
143
144 if (method < 2) {
145     printf("\nThe answers are: \n");
146     for (i = 0; i < row; i++){
147         printf("x_new[%d] = %0.9lf\n", i+1,x_new[i]);
148     }
149     printf("\nTotal Number of Iterations = %d\n", k);
150 }else {
151     printf("% 0.2lf %d \n",w,k);
152 }
153 }

```

```
154
155     return 0;
156 }
157
158
```